Package: SFDesign (via r-universe)

May 20, 2026

Type Package

Title Space-Filling Designs

Version 0.1.5

Date 2026-02-18

Maintainer Shangkun Wang <shangkunwang01@gmail.com>

Description Construct various types of space-filling designs, including Latin hypercube designs, clustering-based designs, maximin designs, maximum projection designs, and uniform designs (Joseph 2016 <doi:10.1080/08982112.2015.1100447>). It also offers the option to optimize designs based on user-defined criteria. This work is supported by U.S. National Science Foundation grant DMS-2310637.

Depends Rcpp (>= 1.0.8)

Imports GenSA, nloptr, primes, proxy, spacefillr

LinkingTo Rcpp, RcppArmadillo

License GPL (>= 2)

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.3.2

Author Shangkun Wang [aut, cre], Roshan Joseph [aut]

Config/pak/sysreqs cmake

Repository https://shangkunwang01.r-universe.dev

Date/Publication 2026-02-18 22:20:02 UTC

RemoteUrl https://github.com/cran/SFDesign

RemoteRef HEAD

RemoteSha d92753a86a8da05ca2dfb4c48ec16423f9005804

Contents

`SFDesign-package` *Space-Filling designs*

Description

This pacakge offers a comprehensive suite of functions to construct various types of space-filling designs, including Latin hypercube designs, clustering-based designs, maximin designs, maximum projection designs, and uniform designs (Joseph 2016). It also offers the option to optimize designs based on user-defined criteria.

Author(s)

Shangkun Wang, V. Roshan Joseph

Maintainer: Shangkun Wang <shangkunwang01@gmail.com>

References

Wang, Shangkun, Xie, Weijun and V. Roshan Joseph. SFDesign: An R package for Space-Filling Designs.

Joseph, V. R. (2016). Space-filling designs for computer experiments: A review. Quality Engineering, 28(1), 28-35.

`cluster.error` *Clustering error*

Description

This function computes the clustering error.

Usage

```
cluster.error(design, X = NULL, alpha = 1)
```

Arguments

`design`	a design matrix.
`X`	candidate points in $[0, 1]^p$. If X is not provided, Sobol points are generated as candidate points.
`alpha`	power of the Euclidean distance.

Details

`cluster.error` computes the clustering error. The clustering error for a design $D = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^T$ is defined as $\frac{1}{N} \sum_{i=1}^{n} \sum_{\boldsymbol{x} \in V_i} \|\boldsymbol{x} - \boldsymbol{x}_i\|^{\alpha}$, where V_i is the Voronoi cell of each design point $\boldsymbol{x}_i$ for $i = 1, \ldots, n$, N is the size of X. When $\alpha = 2$, we obtain K-means and when $\alpha = 1$, we obtain K-medians.

Value

clustering error of the design.

Examples

```
n = 20
p = 3
D = randomLHD(n, p)
cluster.error(D)
```

`clustering.design` *Designs generated by clustering algorithms*

Description

This function is for producing designs by minimizing the clustering error.

Usage

```
clustering.design(
  n,
  p,
  X = NULL,
  D.ini = NULL,
  multi.start = 1,
  alpha = 1,
  Lloyd.iter.max = 100,
  cen.iter.max = 10,
  Lloyd.tol = 1e-04,
  cen.tol = 1e-04
)
```

Arguments

`n`	design size.
`p`	design dimension.
`X`	candidate points in $[0,1]^p$. If X is not provided, Sobol points is generated as cluster points.
`D.ini`	initial design points. If D.ini is not provided, Sobol points are generated as initial design.
`multi.start`	number of starting designs (cluster centers).
`alpha`	power of the Euclidean distance.
`Lloyd.iter.max`	maximum number of iterations for the Lloyd algorithm.
`cen.iter.max`	maximum number of iterations for the center calculation for each cluster.
`Lloyd.tol`	minimum relative change for the Lloyd algorithm to continue.
`cen.tol`	minimum relative change for the center calculation algorithm to continue.

Details

`clustering.design` produces a design by clustering algorithms. It minimize the clustering error (see `cluster.error`) by Lloyd's algorithm. When $\alpha > 2$, accelerated gradient descent is used to find the center for each cluster (Mak, S. and Joseph, V. R. 2018). When $\alpha \leq 2$: Weizfeld algorithm is used to find the center for each cluster. Let $\boldsymbol{x}_i^{(0)} = \boldsymbol{x}_i$ denote the initial position of the ith center and and let $\boldsymbol{S}_i$ represent the points within its Voronoi cell. The center is then updated as: $\boldsymbol{x}_i^{(k+1)} = \left(\sum_{\boldsymbol{s}\in\boldsymbol{S}_i} \frac{\boldsymbol{s}}{\|\boldsymbol{s}-\boldsymbol{x}_i^{(k)}\|_2^{2-\alpha}}\right) \Big/ \left(\sum_{\boldsymbol{s}\in\boldsymbol{S}_i} \frac{1}{\|\boldsymbol{s}-\boldsymbol{x}_i^{(k)}\|_2^{2-\alpha}}\right)$ for $k = 0, 1, \ldots$.

Value

design	final design points.
cluster	cluster assignment for each cluster points.
cluster.error	final cluster error.
total.iter	total number of iterations used in the Lloyd algorithm.
crit.hist	history of clustering error values at each iteration.

References

Mak, S. and Joseph, V. R. (2018), "Minimax and minimax projection designs using clustering," Journal of Computational and Graphical Statistics, 27, 166–178.

Examples

```
# Example 1
n = 10
p = 3
X = spacefillr::generate_sobol_set(1e5*p, p)
D = clustering.design(n, p, X)

# Example 2: multi-start
n = 10
p = 3
X = spacefillr::generate_sobol_set(1e5*p, p)
D = clustering.design(n, p, X, multi.start=2)
```

---

continuous.optim *Continuous optimization of a design*

Description

This function does continuous optimization of an existing design based on a specified criterion. It has an option to run simulated annealing after the continuous optimization.

Usage

```
continuous.optim(
  D.ini,
  objective,
  gradient = NULL,
  iteration = 10,
  sa = FALSE,
  sa.objective = NULL
)
```

Arguments

D.ini	initial design matrix.
objective	the criterion to minimize for the design. It can also return gradient information at the same time in a list with elements "objective" and "gradient".
gradient	the gradient of the objective with respect to the design.
iteration	number iterations for LBFGS.
sa	whether to use simulated annealing. If the final criterion is different from the objective function specified above, simulated annealing can be useful. Use this option only when the design size and dimension are not large.
sa.objective	the criterion to minimize for the simulated annealing.

Details

continuous.optim optimizes an existing design based on a specified criterion. It is a wrapper for the L-BFGS-B function from the nloptr packakge (Johnson 2008) and/or GenSA function in GenSA package (Xiang, Gubian, Suomela and Hoeng 2013).

Value

the optimized design.

References

Johnson, S. G. (2008), The NLopt nonlinear-optimization package, available at https://github.com/stevengj/nlopt.
Xiang Y, Gubian S, Suomela B, Hoeng (2013). "Generalized Simulated Annealing for Efficient Global Optimization: the GenSA Package for R". The R Journal Volume 5/1, June 2013.

Examples

```
# Below is an example showing how to create functions needed to generate MaxPro design manually by
# continuous.optim without using the maxpro.optim function in the package.
compute.distance.matrix <- function(A){
   log_prod_metric = function(x, y) 2 * sum(log(abs(x-y)))
   return (c(proxy::dist(A, log_prod_metric)))
}
optim.obj = function(x){
  D = matrix(x, nrow=n, ncol=p)
  d = exp(compute.distance.matrix(D))
  d_matrix = matrix(0, n, n)
  d_matrix[lower.tri(d_matrix)] = d
  d_matrix = d_matrix + t(d_matrix)
  fn = sum(1/d)
  lfn = log(fn)
  I = diag(n)
  diag(d_matrix) = rep(1,n)
  A = B = D
  for(j in 1:p)
  {
    A = t(outer(D[,j], D[,j], "-"))
```

```
    diag(A) = rep(1, n)
    B[, j] = diag((1/A - I) %*% (1/d_matrix - I))
  }
  grad = 2 * B / fn
  return(list("objective"=lfn, "gradient"=grad))
}
n = 20
p = 3
D.ini = maxproLHD(n, p)$design
D = continuous.optim(D.ini, optim.obj)
```

`customLHD` *Generate a Latin-hypercube design (LHD) based on a custom criterion*

Description

This function generates a LHD by minimizing a user-specified design criterion.

Usage

```
customLHD(
  compute.distance.matrix,
  compute.criterion,
  update.distance.matrix,
  n,
  p,
  design = NULL,
  max.sa.iter = 1e+06,
  temp = 0,
  decay = 0.95,
  no.update.iter.max = 400,
  num.passes = 10,
  max.det.iter = 1e+06,
  method = "full",
  scaled = TRUE
)
```

Arguments

`compute.distance.matrix`
: a function to calculate pairwise distance

`compute.criterion`
: a function to calculate the criterion based on the pairwise distance

`update.distance.matrix`
: a function to update the distance matrix after swapping one column of two design points

n	design size.
p	design dimension.
design	an initial LHD. If design=NULL, a random LHD is generated.
max.sa.iter	maximum number of swapping involved in the simulated annealing (SA) algorithm.
temp	initial temperature of the simulated annealing algorithm. If temp=0, it will be automatically determined.
decay	the temperature decay rate of simulated annealing.
no.update.iter.max	the maximum number of iterations where there is no update to the global optimum before SA stops.
num.passes	the maximum number of passes of the whole design matrix if deterministic swapping is used.
max.det.iter	maximum number of swapping involved in the deterministic swapping algorithm.
method	choice of "deterministic", "sa", or "full". See details for the description of each choice.
scaled	whether the design is scaled to unit hypercube. If scaled=FALSE, the design is represented by integer numbers from 1 to design size. Leave it as TRUE when no initial design is provided.

Details

customLHD generates a LHD by minimizing a user-specified design criterion.

- If method='sa', then a simulated annealing algorithm is used to optimize the LHD. To custom the optimization process, you can change the default values for max.sa.iter, temp, decay, no.update.iter.max. In this optimization step, two design points are randomly chosen and their coordinate along one dimension are swaped. If the new design improves the criterion, then it is accepted; otherwise, it is accepted with some probability.
- If method='deterministic', then a deterministic swap algorithm is used to optimize the LHD. To custom the optimization process, you can change the default values for num.passes, max.det.iter. In this optimization step, we swap the coordinates of all pairs of design points (start with design point 1 with design point 2, then 1 with 3, ... 1 with n, then 2 with 3 until n-1 with n). Only accept the change if the swap leads to an improvement.
- If method='full', then optimization goes through the above two stages.

Value

design	optimized LHD.
total.iter	total number of swaps in the optimization.
criterion	optimized criterion.
crit.hist	criterion history during the optimization process.

Examples

```
# Below is an example showing how to create functions needed to generate
# MaxPro LHD manually by customLHD without using the maxproLHD function in
# the package.
compute.distance.matrix <- function(A){
   s = 2
   log_prod_metric = function(x, y) s * sum(log(abs(x-y)))
   return (c(proxy::dist(A, log_prod_metric)))
}
compute.criterion <- function(n, p, d) {
  s = 2
  dim <- as.integer(n * (n - 1) / 2)
  # Find the minimum distance
  Dmin <- min(d)
  # Compute the exponential summation
  avgdist <- sum(exp(Dmin - d))
  # Apply the logarithmic transformation and scaling
  avgdist <- log(avgdist) - Dmin
  avgdist <- exp((avgdist - log(dim)) * (p * s) ^ (-1))
  return(avgdist)
}

update.distance.matrix <- function(A, col, selrow1, selrow2, d) {
  s = 2
  n = nrow(A)
  # transform from c++ idx to r idx
  selrow1 = selrow1 + 1
  selrow2 = selrow2 + 1
  col = col + 1
  # A is the updated matrix
  row1 <- min(selrow1, selrow2)
  row2 <- max(selrow1, selrow2)

  compute_position <- function(row, h, n) {
    n*(h-1) - h*(h-1)/2 + row-h
  }

  # Update for rows less than row1
  if (row1 > 1) {
    for (h in 1:(row1-1)) {
      position1 <- compute_position(row1, h, n)
      position2 <- compute_position(row2, h, n)
      d[position1] <- d[position1] + s * log(abs(A[row1, col] - A[h, col])) -
          s * log(abs(A[row2, col] - A[h, col]))
      d[position2] <- d[position2] + s * log(abs(A[row2, col] - A[h, col])) -
          s * log(abs(A[row1, col] - A[h, col]))
    }
  }

  # Update for rows between row1 and row2
  if ((row2-row1) > 1){
    for (h in (row1+1):(row2-1)) {
```

```
      position1 <- compute_position(h, row1, n)
      position2 <- compute_position(row2, h, n)
      d[position1] <- d[position1] + s * log(abs(A[row1, col] - A[h, col])) -
          s * log(abs(A[row2, col] - A[h, col]))
      d[position2] <- d[position2] + s * log(abs(A[row2, col] - A[h, col])) -
          s * log(abs(A[row1, col] - A[h, col]))
    }
  }

  # Update for rows greater than row2
  if (row2 < n) {
    for (h in (row2+1):n) {
      position1 <- compute_position(h, row1, n)
      position2 <- compute_position(h, row2, n)
      d[position1] <- d[position1] + s * log(abs(A[row1, col] - A[h, col])) -
          s * log(abs(A[row2, col] - A[h, col]))
      d[position2] <- d[position2] + s * log(abs(A[row2, col] - A[h, col])) -
          s * log(abs(A[row1, col] - A[h, col]))
    }
  }
  return (d)
}

n = 6
p = 2
# Find an appropriate initial temperature
crit1 = 1 / (n-1)
crit2 = (1 / ((n-1)^(p-1) * (n-2))) ^ (1/p)
delta = crit2 - crit1
temp = - delta / log(0.99)
result_custom = customLHD(compute.distance.matrix,
function(d) compute.criterion(n, p, d),
update.distance.matrix, n, p, temp = temp)
```

`full.factorial` *Full factorial design*

Description

This function generates a full factorial design.

Usage

```
full.factorial(p, level)
```

Arguments

`p`	design dimension.
`level`	an integer specifying the number of levels.

Details

`full.factorial` generates a p dimensional full factorial design.

Value

a full factorial design matrix (scaled to [0, 1])

Examples

```
p = 3
level = 3
D = full.factorial(p, level)
```

`maximin.augment` *Augment a design by adding new design points that minimize the reciprocal distance criterion greedily*

Description

This function augments a design by adding new design points one-at-a-time that minimize the reciprocal distance criterion.

Usage

```
maximin.augment(n, p, D.ini, candidate = NULL, r = 2 * p)
```

Arguments

`n`	the size of the final design.
`p`	design dimension.
`D.ini`	initial design.
`candidate`	candidate points to choose from. The default candidates are Sobol points of size 100n.
`r`	the power parameter in the `maximin.crit`. By default it is set as 2p.

Details

`maximin.augment` augments a design by adding new design points that minimize the reciprocal distance criterion (see `maximin.crit`) greedily. In each iteration, the new design points is selected as the one from the candidate points that has the smallest sum of reciprocal distance to the existing design, that is, $\boldsymbol{x}_{k+1} = \arg\min_{\boldsymbol{x}} \sum_{i=1}^{k} \frac{1}{\|\boldsymbol{x}-\boldsymbol{x}_i\|^r}$.

Value

the augmented design.

Examples

```
# Example 1
n.ini = 10
n = 20
p = 3
D.ini = maximinLHD(n.ini, p)$design
D = maximin.augment(n, p, D.ini)

# Example 2: augment from given candidates
n.ini = 10
n = 10
p = 5
D.ini = maximinLHD(n.ini, p)$design
candidates = matrix(runif(1000), ncol=p)
D = maximin.augment(n, p, D.ini, candidates)
```

`maximin.crit` *Maximin criterion*

Description

This function calculates the maximin distance or the average reciprocal distance of a design.

Usage

```
maximin.crit(design, r = 2 * ncol(design), surrogate = FALSE)
```

Arguments

`design`	the design matrix.
`r`	the power used in the reciprocal distance objective function. The default value is set as twice the dimension of the design.
`surrogate`	whether to return the surrogate average reciprocal distance objective function or the maximin distance. If setting surrogate=TRUE, then the average reciprocal distance is returned.

Details

`maximin.crit` calculates the maximin distance or the average reciprocal distance of a design. The maximin distance for a design $D = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^T$ is defined as $\phi_{Mm} = \min_{i \neq j} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$. In optimization, the average reciprocal distance is usually used (Morris and Mitchell, 1995):

$$\phi_{\text{rec}} = \left(\frac{2}{n(n-1)} \sum_{i \neq j} \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^r} \right)^{1/r}.$$

The r is a power parameter and when it is large enough, the reciprocal distance is similar to the exact maximin distance.

Value

the maximin distance or reciprocal distance of the design.

References

Morris, M. D. and Mitchell, T. J. (1995), "Exploratory designs for computational experiments," Journal of statistical planning and inference, 43, 381–402.

Examples

```
n = 20
p = 3
D = randomLHD(n, p)
maximin.crit(D)
```

`maximin.optim` *Optimize a design based on maximin or reciprocal distance criterion*

Description

This function optimizes a design by continuous optimization based on reciprocal distance criterion. A simulated annealing step can be enabled in the end to directly optimize the maximin distance criterion.

Usage

```
maximin.optim(D.ini, iteration = 10, sa = FALSE, find.best.ini = FALSE)
```

Arguments

`D.ini`	the initial design.
`iteration`	number iterations for L-BFGS-B algorithm.
`sa`	whether to use simulated annealing in the end. If sa=TRUE, continuous optimization is first used to optimize the reciprocal distance criterion and then SA is performed to optimize the maximin criterion.
`find.best.ini`	whether to generate other initial designs. If find.best.ini=TRUE, it will first find the closest full factorial design in terms of size to `D.ini`. If the size of the full factorial design is larger than `D.ini`, design points will be removed by the `maximin.remove` function. If the the size of the full factorial design is smaller than `D.ini`, then we will augment the design by `maximin.augment`. In this case, we have two different candidate set to choose from: one is a full factorial design with level+1 and another is Sobol points. All initial designs are optimized and the best is returned.

Details

`maximin.optim` optimizes a design by L-BFGS-B algorithm (Liu and Nocedal 1989) based on the reciprocal distance criterion. A simulated annealing step can be enabled in the end to directly optimize the maximin distance criterion. Optimization detail can be found in `continuous.optim`. We also provide the option to try other initial designs generated internally besides the `D.ini` provided by the user (see argument `find.best.ini`).

Value

`design`	the optimized design.
`D.ini`	initial designs. If find.best.ini=TRUE, a list will be returned containing all the initial designs considered.

References

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. Mathematical programming, 45(1), 503-528.

Examples

```
n = 20
p = 3
D = maximinLHD(n, p)$design
D = maximin.optim(D, sa=FALSE)$design
# D = maximin.optim(D, sa=TRUE)$design # Let sa=TRUE only when the n and p is not large.
```

`maximin.remove` *Sequentially remove design points from a design while maintaining low reciprocal distance criterion as possible*

Description

This function sequentially removes design points one-at-a-time from a design while maintaining low reciprocal distance criterion as possible.

Usage

```
maximin.remove(D, n.remove, r = 2 * p)
```

Arguments

`D`	the design matrix.
`n.remove`	number of design points to remove.
`r`	the power parameter in the `maximin.crit`. By default it is set as 2p.

Details

maximin.remove sequentially removes design points from a design in a greedy way while maintaining low reciprocal distance criterion (see maximin.crit) as possible. In each iteration, the design point with the largest sum of reciprocal distances with the other design points is removed, that is, $k^* = \arg\max_k \sum_{i \neq k} \frac{1}{\|\boldsymbol{x}_k - \boldsymbol{x}_i\|^r}$.

Value

the updated design.

Examples

```
# Example 1
n = 20
p = 3
n.remove =  5
D = maximinLHD(n, p)$design
D = maximin.remove(D, n.remove)

# Example 2 : generate maximin design from candidates
N = 500
n = 20
p = 2
candidates = matrix(runif(N*p), ncol=p)
D = maximin.remove(candidates, N-n)
```

---

maximinLHD *Generate a maximin Latin-hypercube design (LHD)*

Description

This function generates a LHD with large maximin distance.

Usage

```
maximinLHD(
  n,
  p,
  design = NULL,
  power = 2 * p,
  max.sa.iter = 1e+06,
  temp = 0,
  decay = 0.95,
  no.update.iter.max = 100,
  num.passes = 10,
  max.det.iter = 1e+06,
  method = "full",
```

```
  scaled = TRUE
)
```

Arguments

n	design size.
p	design dimension.
design	an initial LHD. If design=NULL, a random LHD is generated.
power	the power used in the maximin objective function.
max.sa.iter	maximum number of swapping involved in the simulated annealing (SA) algorithm.
temp	initial temperature of the simulated annealing algorithm. If temp=0, it will be automatically determined.
decay	the temperature decay rate of simulated annealing.
no.update.iter.max	the maximum number of iterations where there is no update to the global optimum before SA stops.
num.passes	the maximum number of passes of the whole design matrix if deterministic swapping is used.
max.det.iter	maximum number of swapping involved in the deterministic swapping algorithm.
method	choice of "deterministic", "sa", or "full". If the method="full", the design is first optimized by SA and then deterministic swapping.
scaled	whether the design is scaled to unit hypercube. If scaled=FALSE, the design is represented by integer numbers from 1 to design size. Leave it as TRUE when no initial design is provided.

Details

maximinLHD generates a LHD or optimize an existing LHD to achieve large maximin distance by optimizing the reciprocal distance (see maximin.crit). The optimization details can be found in customLHD.

Value

design	final design points.
total.iter	total number of swaps in the optimization.
criterion	final optimized criterion.
crit.hist	criterion history during the optimization process.

Examples

```
# We show three different ways to use this function.
n = 20
p = 3
D.random = randomLHD(n, p)
# optimize over a random LHD by SA
D = maximinLHD(n, p, D.random, method='sa')
# optimize over a random LHD by deterministic swapping
D = maximinLHD(n, p, D.random, method='deterministic')
# Directly generate an optimized LHD for maximin criterion which goes
# through the above two optimization stages.
D = maximinLHD(n, p)
```

`maxpro.crit` *Maximum projection (MaxPro) criterion*

Description

This function calculates the MaxPro criterion of a design.

Usage

```
maxpro.crit(design, delta = 0)
```

Arguments

`design`	the design matrix.
`delta`	a small value added to the denominator of the maximum projection criterion. By default it is set as zero.

Details

`maxpro.crit` calculates the MaxPro criterion of a design. The MaxPro criterion for a design $D = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^T$ is defined as

$$\left\{ \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\prod_{l=1}^{p} [(x_{il} - x_{jl})^2 + \delta]} \right\}^{1/p},$$

where p is the dimension of the design (Joseph, V. R., Gul, E., & Ba, S. 2015).

Value

the MaxPro criterion of the design.

References

Joseph, V. R., Gul, E., & Ba, S. (2015). Maximum projection designs for computer experiments. Biometrika, 102(2), 371-380.

Examples

```
n = 20
p = 3
D = randomLHD(n, p)
maxpro.crit(D)
maxpro.crit(D, 1e-4) # add a nugget term for stability
```

`maxpro.optim` *Optimize a design based on the maximum projection criterion*

Description

This function optimizes a design by continuous optimization based on maximum projection criterion (Joseph, V. R., Gul, E., & Ba, S. 2015).

Usage

```
maxpro.optim(D.ini, iteration = 10)
```

Arguments

`D.ini`	the initial design.
`iteration`	number iterations for L-BFGS-B.

Details

`maxpro.optim` optimizes a design by L-BFGS-B algorithm (Liu and Nocedal 1989) based on the maximum projection criterion `maxpro.crit`.

Value

`design`	optimized design.
`D.ini`	initial design.

References

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. Mathematical programming, 45(1), 503-528.

Joseph, V. R., Gul, E., & Ba, S. (2015). Maximum projection designs for computer experiments. Biometrika, 102(2), 371-380.

Examples

```
n = 20
p = 3
D = maxproLHD(n, p)$design
D = maxpro.optim(D)$design
```

`maxpro.remove` *Sequentially remove design points from a design while maintaining low maximum projection criterion as possible*

Description

This function sequentially removes design points one-at-a-time from a design while maintaining low maximum projection criterion as possible.

Usage

```
maxpro.remove(D, n.remove, delta = 0)
```

Arguments

`D`	design
`n.remove`	number of design points to remove
`delta`	a small value added to the denominator of the maximum projection criterion. By default it is set as zero.

Details

`maxpro.remove` sequentially removes design points from a design while maintaining low maximum projection criterion (see `maxpro.crit`) as possible. The maximum projection criterion is modified to include a small delta term:

$$\phi_{\text{maxpro}}(\boldsymbol{D}_n) = \left\{ \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\prod_{l=1}^{p}(x_{il} - x_{jl})^2 + \delta} \right\}^{1/p}.$$

The index of the point to remove is $k^* = \arg\min_k \sum_{i \neq k} \frac{1}{\prod_{l=1}^{p}(x_{il} - x_{kl})^2 + \delta}$.

Value

the updated design.

Examples

```
# Example 1
n = 20
p = 3
n.remove =  5
D = maxproLHD(n, p)$design
D = maxpro.remove(D, n.remove)

# Example 2 : generate maxpro design from candidates
N = 500
n = 20
p = 2
candidates = matrix(runif(N*2), ncol=2)
D = maxpro.remove(candidates, N-n)
```

---

maxproLHD *Generate a MaxPro Latin-hypercube design*

Description

This function generates a MaxPro Latin-hypercube design.

Usage

```
maxproLHD(
  n,
  p,
  design = NULL,
  max.sa.iter = 1e+06,
  temp = 0,
  decay = 0.95,
  no.update.iter.max = 400,
  num.passes = 10,
  max.det.iter = 1e+06,
  method = "full",
  scaled = TRUE
)
```

Arguments

n	design size.
p	design dimension.
design	an initial LHD. If design=NULL, a random LHD is generated.
max.sa.iter	maximum number of swapping involved in the simulated annealing (SA) algorithm.

temp	initial temperature of the simulated annealing algorithm. If temp=0, it will be automatically determined.
decay	the temperature decay rate of simulated annealing.
no.update.iter.max	the maximum number of iterations where there is no update to the global optimum before SA stops.
num.passes	the maximum number of passes of the whole design matrix if deterministic swapping is used.
max.det.iter	maximum number of swapping involved in the deterministic swapping algorithm.
method	choice of "deterministic", "sa", or "full". If the method="full", the design is first optimized by SA and then deterministic swapping.
scaled	whether the design is scaled to unit hypercube. If scaled=FALSE, the design is represented by integer numbers from 1 to design size. Leave it as TRUE when no initial design is provided.

Details

maxproLHD generates a MaxPro Latin-hypercube design (Joseph, V. R., Gul, E., & Ba, S. 2015). The major difference with the MaxPro packages is that we have a deterministic swap algorithm, which can be enabled by setting method="deterministic" or method="full". For optimization details, see the detail section in customLHD.

Value

design	final design points.
total.iter	total number of swaps in the optimization.
criterion	final optimized criterion.
crit.hist	criterion history during the optimization process.

References

Joseph, V. R., Gul, E., & Ba, S. (2015). Maximum projection designs for computer experiments. Biometrika, 102(2), 371-380.

Examples

```
# Example 1
n = 20
p = 3
# optimize by SA
D = maxproLHD(n, p, method='sa')
# optimize by deterministic swapping
D = maxproLHD(n, p, method='deterministic')
# generate an optimized LHD for maxpro criterion which goes
# through the above two optimization stages.
D = maxproLHD(n, p)
```

randomLHD *Random Latin hypercube design*

Description

This function generates a random Latin hypercube design.

Usage

```
randomLHD(n, p)
```

Arguments

n	design size.
p	design dimension.

Details

randomLHD generates a random Latin hypercube design.

Value

a random Latin hypercube design.

Examples

```
n = 20
p = 3
D = randomLHD(n, p)
```

uniform.crit *Uniform criterion*

Description

This function calculates the wrap-around discrepancy of a design.

Usage

```
uniform.crit(design)
```

Arguments

design	a design matrix.

Details

`uniform.crit` calculates the wrap-around discrepancy of a design. The wrap-around discrepancy for a design $D = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^T$ is defined as (Hickernell, 1998):

$$\phi_{wa} = -\left(\frac{4}{3}\right)^p + \frac{1}{n^2}\sum_{i,j=1}^{n}\prod_{k=1}^{p}\left[\frac{3}{2} - |x_{ik} - x_{jk}|(1 - |x_{ik} - x_{jk}|)\right].$$

Value

wrap-around discrepancy of the design

References

Hickernell, F. (1998), "A generalized discrepancy and quadrature error bound," Mathematics of computation, 67, 299–322.

Examples

```
n = 20
p = 3
D = randomLHD(n, p)
uniform.crit(D)
```

`uniform.discrete` *Generate a uniform design for discrete factors with different number of levels*

Description

This function generates a uniform design for discrete factors with different number of levels.

Usage

```
uniform.discrete(
  t,
  p,
  levels,
  design = NULL,
  max.sa.iter = 1e+06,
  temp = 0,
  decay = 0.95,
  no.update.iter.max = 400,
  num.passes = 10,
  max.det.iter = 1e+06,
  method = "full",
  scaled = TRUE
)
```

Arguments

t	multiple of the least common multiple of the levels.
p	design dimension.
levels	a vector of the number of levels for each dimension.
design	an initial design. If design=NULL, a random design is generated.
max.sa.iter	maximum number of swapping involved in the simulated annealing (SA) algorithm.
temp	initial temperature of the simulated annealing algorithm. If temp=0, it will be automatically determined.
decay	the temperature decay rate of simulated annealing.
no.update.iter.max	the maximum number of iterations where there is no update to the global optimum before SA stops.
num.passes	the maximum number of passes of the whole design matrix if deterministic swapping is used.
max.det.iter	maximum number of swapping involved in the deterministic swapping algorithm.
method	choice of "deterministic", "sa", or "full". If the method="full", the design is first optimized by SA and then deterministic swapping.
scaled	whether the design is scaled to unit hypercube. If scaled=FALSE, the design is represented by integer numbers from 1 to design size. Leave it as TRUE when no initial design is provided.

Details

`uniform.discrete` generates a uniform design of discrete factors with different number of levels by minimizing the wrap-around discrepancy criterion (see `uniform.crit`).

Value

design	final design points.
design.int	design transformed to integer numbers for each dimenion
total.iter	total number of swaps in the optimization.
criterion	final optimized criterion.
crit.hist	criterion history during the optimization process.

Examples

```
p = 5
levels = c(3, 4, 6, 2, 3)
t = 1
D = uniform.discrete(t, p, levels)
```

`uniform.optim` *Optimize a design based on the wrap-around discrepancy*

Description

This function optimizes a design through continuous optimization of the wrap-around discrepancy.

Usage

```
uniform.optim(D.ini, iteration = 10)
```

Arguments

`D.ini`	the initial design.
`iteration`	number iterations for LBFGS.

Details

`uniform.optim` optimizes a design through continuous optimization of the wrap-around discrepancy (see `uniform.crit`) by L-BFGS-B algorithm (Liu and Nocedal 1989).

Value

`design`	optimized design.
`D.ini`	initial design.

References

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. Mathematical programming, 45(1), 503-528.

Examples

```
n = 20
p = 3
D = uniformLHD(n, p)$design
D = uniform.optim(D)$design
```

`uniformLHD` *Generate a uniform Latin-hypercube design (LHD)*

Description

This function generates a uniform LHD by minimizing the wrap-around discrepancy.

Usage

```
uniformLHD(
  n,
  p,
  design = NULL,
  max.sa.iter = 1e+06,
  temp = 0,
  decay = 0.95,
  no.update.iter.max = 400,
  num.passes = 10,
  max.det.iter = 1e+06,
  method = "full",
  scaled = TRUE
)
```

Arguments

`n`	design size.
`p`	design dimension.
`design`	an initial LHD. If design=NULL, a random LHD is generated.
`max.sa.iter`	maximum number of swapping involved in the simulated annealing (SA) algorithm.
`temp`	initial temperature of the simulated annealing algorithm. If temp=0, it will be automatically determined.
`decay`	the temperature decay rate of simulated annealing.
`no.update.iter.max`	the maximum number of iterations where there is no update to the global optimum before SA stops.
`num.passes`	the maximum number of passes of the whole design matrix if deterministic swapping is used.
`max.det.iter`	maximum number of swapping involved in the deterministic swapping algorithm.
`method`	choice of "deterministic", "sa", or "full". If the method="full", the design is first optimized by SA and then deterministic swapping.
`scaled`	whether the design is scaled to unit hypercube. If scaled=FALSE, the design is represented by integer numbers from 1 to design size. Leave it as TRUE when no initial design is provided.

Details

uniformLHD generates a uniform LHD minimizing wrap-around discrepancy (see uniform.crit). The optimization details can be found in customLHD.

Value

design	final design points.
total.iter	total number of swaps in the optimization.
criterion	final optimized criterion.
crit.hist	criterion history during the optimization process.

Examples

```
n = 20
p = 3
# optimize by SA
D = uniformLHD(n, p, method='sa')
# optimize by deterministic swapping
D = uniformLHD(n, p, method='deterministic')
# generate an optimized LHD for wrap-around discrepancy which goes
# through the above two optimization stages.
D = uniformLHD(n, p)
```

Index